IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of: **Levine et al.**	§	
	§	Group Art Unit: **2183**
Serial No. **10/674,606**	§	
	§	Examiner: **Fiegle, Ryan Paul**
Filed: **September 30, 2003**	§	
	§	
For: **Method and Apparatus to Autonomically Generate Information on Calls and Returns in a Program**	§	

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

35525
PATENT TRADEMARK OFFICE
CUSTOMER NUMBER

REPLY BRIEF (37 C.F.R. 41.41)

This Reply Brief is submitted in response to the Examiner's Answer mailed on January 11, 2007.

No fees are believed to be required to file a Reply Brief. If any fees are required, I authorize the Commissioner to charge these fees which may be required to IBM Corporation Deposit Account No. 09-0447.

RESPONSE TO EXAMINER'S ANSWER

I. Claims 1, 8, 15, and 22-24

I.A. The Examiner Fails to Present a *Prima Facie* Case of Obviousness Because the References do not Teach or Suggest all the Features of Claims 1, 8, 15, and 22-24.

In response to the *Appeal Brief*, the Examiner states the following:

> **(10) Response to Argument**
>
> The appellant has made the following argument on page 10 of the appeal brief:
>
> "*Smolders* teaches generating an interrupt after each branch instruction. The Office Action asserts that this method would include calls and returns. Based on the assertion, the Office Action concludes that *Smolders* discloses the claimed feature of, "storing a respective indicator ... associated with each call and return in the program". However, this interpretation ignores the differences between a branch and a call or return."
>
> A call is just a specific word that refers to a branch that branches into a subroutine. Likewise, a return is a branch that branches out of a subroutine.
>
> High-Tech Dictionary from computeruser.com defines unconditional branch as, "a computer program instruction that transfers control of a different part of the program without requiring a decision."
>
> FOLDOC, the free online dictionary of computing defines jump as "(Or "branch") The term for a goto instruction, usually in a context of machine languages. "Branch" may be synonymous with "jump", or may refer to jumps that depend on a condition."
>
> Webster's New World Computer Dictionary, 9th Ed., defines branch as "3. In programming, a type of control structure that routes program execution to a subroutine."
>
> Therefore, calls and returns, both, are in fact branches, though special kinds of branches. However, some, such as Webster's New-World Computer Dictionary, explicitly consider a branch to be a call. The appellant argues on pages 11 and 12 of the appeal brief that calls and returns differ from branch instructions because calls and returns infer state management and address saving. However, it should be noted that there are some instruction set architectures that do not have dedicated call and return instructions that do this; it is required to do it manually. In addition,

there are no set rules for what constitutes a call or return. Many architectures have differing calling conventions, meaning that sometimes it is up to the caller to save the state while sometimes it is the callee's responsibility. Further, the appellant did not disclose any explicit definitions for either call or return within the specification.

Smolders does not explicitly state that his method is used for calls and returns, but merely branches (column 4, lines 15-18). However, since it has been shown that calls and returns are unquestionably types of branches, it would have been obvious to one of ordinary skill in the pertinent art that *Smolder's* method would be applicable to calls and returns. Little, if any, modifications would need to be made to the present implementation. It should also be noted that each branch in *Smolders* causes an interrupt to collect the data. By definition, an interrupt will make a call to a subroutine, in this case to collect the data, and then return to where the program was interrupted. The claim limitation in question states, "storing an indicator in the indicator location associated with each call and return in the program." A broad and reasonable interpretation of the claim would lead one of ordinary skill in the art to come to the conclusion that since each branch in *Smolders* will cause a call and return associated with it, and the data generation within the subroutine will be associated with these calls and returns, *Smolders* interrupt caused by encountering the branch will fulfill the limitation of "storing an indicator in the indicator location associated with each call and return in the program" in addition to the reasons stated above.

Examiner's Answer, dated January 11, 2007, pp. 9-10.

The Examiner errs in asserting that *Smolders* teaches the features claim 1. Claim 1 is as follows:

> 1. A method in a data processing system for monitoring the execution of a program, the method comprising:
>
> in a system having an indicator location associated with each instruction, storing a respective indicator in the indicator location associated with each call and return in the program; and
>
> executing the program using a processor, wherein the respective indicators associated with the calls and returns cause the processor executing the instructions to generate data on calls and returns in the program.

Smolders does not teach or suggest the feature "**storing** a **respective** indicator **in the indicator location** *associated with each call and return in the program*" as recited in claim 1. The Examiner admits that *Smolders* fails to teach a system having **an indicator location**

associated with each instruction (Examiner's Answer, p. 4). Thus, stating that *Smolders* teaches storing a respective indicator in **the** indicator location is illogical, because *Smolders* lacks the respective indicator location associated with each instruction.

Furthermore, *Smolders* does not teach that the indicator location **stores** any indicators. *Smolders* discloses that a performance monitor feature within a data processing system is programmed to generate a trace interrupt after each branch instruction, or at the end of each basic block of code from a currently running program or process (*See*, Abstract). Even if, *arguendo*, a branch instruction as taught by *Smolders* teaches calls and returns, *Smolders* does not teach storing a respective indicator to the branch instruction in an indicator location associated with each branch in the program. In fact, *Smolders* states "if the current process is a process to be traced, the tracing information is **stored in a trace buffer** for post-processing analysis." (*See*, Abstract, Claim 1, and col. 5, lines 16-23).

Furthermore, claim 1 recites, *inter alia*, "storing a **respective** indicator ... *associated with each call and return in the program*". The term **respective** means that the indicator stored in the indicator location differentiates between a call and a return. Accordingly, because *Smolders* discloses generating an interrupt after every branch without differentiating whether the branch is a call, a return, or some other conditional branch, *Smolders* does not teach that it stores a **respective** indicator ... *associated with each call and return in the program*" as recited in claim 1. Thus, the Examiner errs in asserting that *Smolders* teaches the feature "**storing a respective** indicator **in the indicator location** *associated with each call and return in the program*" for at least the reasons stated above.

Furthermore, the combination of *Smolders* and *Buser* does not teach or suggest the step of "executing the program using a processor, wherein the respective indicators associated with the calls and returns cause the processor executing the instructions to generate data on calls and returns in the program". The information generated by the method recited in claim 1 is specifically "data on calls and returns;" the information is not data generation that is triggered by a call or return. In contrast, *Smolders* is only capable of counting events that occur between branch points. Exemplary data collected by *Smolders* is disclosed to be "counting the number of cycles during a selected executing process or the number of load/store misses occurring within an L2 cache" (*Smolders*, column 3, lines 41-44). Thus, *Smolders* does not disclose a capability to provide "*data on calls and returns*", as recited in Claim 1. *Buser* also does not disclose

generating the claimed "*data on calls and returns*", nor does the rejection assert that *Buser* teaches this claimed feature. Therefore, neither *Smolders*, *Buser*, or a combination of the two references teach or suggest the claimed feature of, "wherein the respective indicators associated with the calls and returns cause the processor executing the instructions to generate data on calls and returns in the program," as in claim 1.

The Examiner rebuts the above argument by stating:

> The appellant has made the following argument on page 12 of the appeal brief: "The information generated by the method recited in Claim 1 is specifically 'data on calls and returns;' the information is not data generation that is triggered by a call or return."
>
> The appellant is using the following interpretation: "...executing the instructions to generate data about calls and returns in the program." On the other hand, the examiner has taken the interpretation, "...executing the instructions to generate data upon encountering calls and returns in the program." Both are acceptable interpretations of the use of "on" within the claim. Neither interpretation is precluded by the remainder of the claims or explicit excerpts from the specification.
>
> It should further be noted that the secondary reference of Subrahmanyam, used in the rejection of other claims, explicitly teaches generating data about calls and returns.

Examiner's Answer, dated January 11, 2007, p. 11.

The Examiner errs in interpreting the language of claim 1. Claim 1 specifically states "wherein the respective indicators associated with the calls and returns **cause** the processor executing the instructions to generate data on calls and returns in the program." The plain language of claim 1 provides that the **respective indicators** associated with the calls and returns cause the generation of data. Thus, the Examiner errs in interpreting that data is generated upon encountering calls and returns in the program.

Additionally, the Examiner errs in asserting that *Subrahmanyam* explicitly teaches generating data about calls and returns. Appellants are unable to locate where *Subrahmanyam* explicitly teaches generating data about calls and returns. If the Examiner continues to insist that *Subrahmanyam* explicitly teaches generating data about calls and returns, then Appellants request that the Examiner cite to the specific portion of *Subrahmanyam* that teaches generating data about calls and returns. Thus, the Examiner errs in asserting that *Smolder*, *Buser*, or the combination thereof, teaches the feature "executing the program using a processor, wherein the

respective indicators associated with the calls and returns cause the processor executing the instructions to generate data on calls and returns in the program" for at least the reasons stated above. Thus, the Examiner fails to state a *prima facie* case of obviousness against claim 1.

I.B. The Examiner Fails to State a Proper Teaching, Suggestion, or Motivation to Combine the References to Achieve the Invention of the Claims.

In response to the *Appeal Brief*, the Examiner states the following:

> The appellant has made the following argument on page 16 of the appeal brief: "Because the references address completely distinct problems, one of ordinary skill would have no reason to combine or otherwise modify the references to achieve the invention of Claim 1."
>
> Both Smolders and Buser have classifications to 712/227, specialized instruction processing in support of testing, debugging and emulating. Further, proper motivation has been shown within the rejection that it would be beneficial to apply Buser to Smolders since there is no other way for multiple processors to see the indicators of Smolders in a shared memory system. As stated above, the implementation and benefits of a shared memory system are well known and therefore one of ordinary skill would readily be motivated to combine Buser to Smolders.

Examiner's Answer, dated January 11, 2007, p. 12.

The Examiner fails to state a proper teaching, suggestion, or motivation based on the references to combine *Smolders* and *Buser*. *Smolders* provides no explicit or implicit teaching or suggestion in regard to a shared memory system. The Examiner states "it would have been obvious for one to combine *Buser*, which deals with the issue of shared memory system, with that of *Smolders* because shared memory systems were well known at the time". However, even if, *arguendo*, shared memory systems were well known at the time, there is no teaching or suggestion within *Smolders* to motivate one of ordinary skill in the art to look at the teachings of *Buser* to combine the references. Additionally, no such teaching existed in the general knowledge of one of ordinary skill in the art at the time of the present invention. Instead, the Examiner has only put forth a proposed advantage to combining the references. However, an advantage is not necessarily a teaching, suggestion, or motivation.

Furthermore, the Examiner errs in stating that applying *Buser* to *Smolders* would be beneficial because there is no other way for multiple processors to see the indicators of *Smolders* in a shared memory system. *Smolders* teaches that a data processing system is programmed to

generate a trace interrupt after each branch instruction. On the other hand, *Buser* discloses a method in which multiple processors must honor a breakpoint in shared memory using a halt indicator. The purpose of *Buser* is that all processors that access shared memory must honor the breakpoint, or else an error will occur. However, even if, *Smolders* was implemented in a multiprocessor environment, the method taught by *Smolders* would not be affected. In fact, *Smolders* specifically states "Although only one processor unit is depicted in the exemplary embodiment, those skilled in the art will appreciate that additional processor units may be utilized in a multiprocessor data processing system in accordance with the present invention." (*Smolders*, col. 2, line 65 – col. 3, line 2). Thus, because the Examiner states that shared memory was well known at the time of *Smolders*, *Smolders*' implementation of a multiprocessor unit would have accounted for the concerns of shared memory. Thus, *Smolders* explicitly excludes any suggestion or motivation to look elsewhere in the event that multiple processor units are employed. Thus, the Examiner fails to state a *prima facie* case of obviousness against claim 1.

II. CONCLUSION

The rejections are clearly in error at least for the reasons stated above. Therefore, Appellants request that the Board of Patent Appeals and Interferences overturn the rejections. Appellants further request that the Board direct the Examiner to allow the claims.

/Theodore D. Fay III/
Theodore D. Fay III
Reg. No. 48,504
YEE & ASSOCIATES, P.C.
PO Box 802333
Dallas, TX 75380
(972) 385-8777

TF/NH